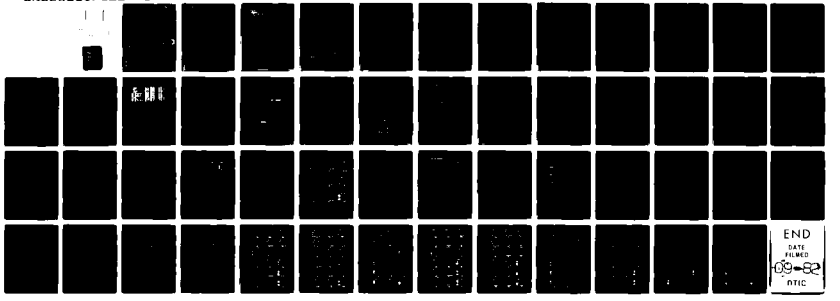
AD-A118 289 STANFORD UNIV CA COMPUTER SYSTEMS LAB F/G 9/2
SOFTWARE RELATED FAILURES ON THE IBM 3081: A RELATIONSHIP WITH --ETC(U)
JUN 82 D J ROSSETTI, R K IYER DAAG29-82-K-0105
UNCLASSIFIED CSL-TN-209 ARO-18690.2-EL NL



ARO 18690.2-EL

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CRC Technical Rpt. 82-8	2. GOVT ACCESSION NO. AD A118289
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SOFTWARE RELATED FAILURES ON THE IBM 3081: A RELATIONSHIP WITH SYSTEM UTILIZATION	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): David J. Rossetti and Ravishankar K. Iyer	8. CONTRACT OR GRANT NUMBER(s): ARO DAAG-29-82-K-0105.
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Center for Reliable Computing, Computer Systems Laboratory, Stanford University, Stanford CA 94305	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: DD Form 2222, Project No. P-18690-EL
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: June 1982 13. NUMBER OF PAGES: 46
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Dr. William A. Sander, Electronics Div., U.S. Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for unlimited public release; distribution unlimited.

DTIC ELECTE AUG 17 1982 F

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official department of the army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

software reliability, workload, statistical failure models, data analysis.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper presents an analysis of software-related system failures on the IBM 3081 at SLAC. We find three broad categories of failures: error handling, control or logic problems and hardware-related. A statistical analysis shows (not unexpectedly) a decreasing failure rate with time. This is especially true in the early part of the study. Not withstanding the decreasing failure with time, we find that the occurrence of failures is strongly correlated with the type and level of (over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

workload prior to the occurrence of a failure. For example, it is shown that the risk of a software-related failure increases in a non-linear fashion with the amount of interactive processing, as measured by paging rate and system overhead. The paper employs a statistical model to describe the load dependency and offers explanations for the observed phenomenon.

Center for Reliable Computing

SOFTWARE RELATED FAILURES ON THE IBM 3081: A RELATIONSHIP WITH SYSTEM UTILIZATION

David J. Rossetti and Ravishankar K. Iyer

CRC Technical Report No. 82-8

(CSL TN No. 209)

June 1982

CENTER FOR RELIABLE COMPUTING
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305 U.S.A.

This work was supported in part by the U. S. Army Research Office under contract number DAAG29-82-K-0105, and by the Department of Energy under contract number DE-AC03-76F00515.

Analysis of Software Related Failures on the IBM 3081: Relationship with System Utilization

David J. Rossetti and Ravishankar K. Iyer

CRC Technical Report No. 82-8

(CSL TN No. 209)

June 1982

CENTER FOR RELIABLE COMPUTING
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305 U.S.A.

ABSTRACT

This paper presents an analysis of software related system failures on the IBM 3081 at SLAC. We find three broad categories of failures: error handling, control or logic problems and hardware-related. A statistical analysis shows (not unexpectedly) a decreasing failure rate with time. This is especially true in the early part of the study. Not withstanding the decreasing failure rate with time, we find that the occurrence of failures is strongly correlated with the type and level of workload prior to the occurrence of a failure. For example, it is shown that the risk of a software related failure increases in a non-linear fashion with the amount of interactive processing, as measured by paging rate and system overhead. The paper employs a statistical model to describe the load dependency and offers explanations for the observed phenomenon.

Keywords: software reliability, workload, statistical failure models, data analysis.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

The highly interactive and diverse nature of modern day systems has made high reliability a central issue in computer system design. Most researchers in the area would agree that it is not feasible to guarantee a perfect system, either in hardware or in software. Accordingly, depending on the nature of the application, it is important to design into the system the ability either to continue operation in the event of a failure or to react to a failure in a predictable manner.

Designing hardware systems that tolerate faults is relatively well understood, at least from a theoretical viewpoint. However, the problem of software fault tolerance (especially the question of hardware/software interaction) has yet to be well understood [Hecht 80]. A reason for this is that neither the error generation process nor the prediction problem are easy to comprehend, although the SIFT studies have been an important contribution [Wensley 78] [Milliar-Smith 81].

Theoretical models can only deal with a restricted class of problems. Most often it is the problems outside the range of theoretical models which cause the most severe malfunctions. Accordingly, at this stage there is no better substitute for results based on actual measurements and experimentation [Curtis 80]; such results are few and far between [Denning 80].

This paper presents results of one such analysis conducted on the VM/370 operating system on the IBM 3081 at the Stanford Linear Accelerator Center (SLAC) computation facility.

The Stanford Linear Accelerator Center is engaged in the study of high energy particle physics. A two mile long linear accelerator and

the associated real-time data network provide a vast amount of physical data for analysis. The SLAC installation (which was reconfigured in February 1981) consists of three processors. There are two IBM 370/168 processors, running the OS/VS2(SVS) operating system, that provide mainly batch service. The last is an IBM 3081 exclusively running the VM/370 (Virtual Machine) operating system. During a typical day, the 3081 complex has approximately 150 time-sharing users with a sizeable compute-bound background load. Although there is some communication with the older system, for practical purposes the 3081 is run as an independent system.

Our general objective was to study the causes of system failures, due to software, in a fully operational production environment. We intended not only to investigate the effect of persistent bugs in reasonably mature software systems, but also to study the interactions with the rest of the system. In particular we wished to consider the following questions:

1. What are the most common types of software related failures and their relative frequencies?
2. Are there any identifiable failure patterns that occur most often? For example, is an inadequate hardware-software interface a frequent cause of system failure?
3. Is there a relationship between operating system failures and the usage environment as represented by various measures of system activity?
4. What inferences can be drawn from our analysis in relation to both the design and testing of large software systems?

Our general approach is to assume no model a priori, but rather to start with a substantial amount of high quality data on software related failures and system activity. We report on statistical trends and relationships found in the data with the aim of discovering an underlying model. The experience gained and the models found would, in our view, provide valuable insight into the question of fault-tolerant design in general and software design in particular. The next section describes the basis for this work in detail and places related studies in perspective.

2. BASIS AND PERSPECTIVE

The term "software reliability model" is usually taken to mean mathematical models for assessing the reliability of software (in terms of statistical parameters such as MTBF)[1] during the development, debugging or testing phases, although a few of these models have also been applied in follow-up operational phases. In this context there have been two distinct approaches to studying software reliability. In the first, software reliability is defined in a manner similar to hardware reliability. Several competing models have appeared in the literature [Musa 1980], and a number of authors have attempted to analyze their suitability; an appreciation of the extent and nature of this discussion can be obtained from [Goel 80]. There is, perhaps, some evidence to suggest that the hardware analogy may have been carried too far [Littlewood 80].

The second approach attempts to exploit the close relationship between software quality measures (e.g. complexity) and reliability. The parameters of these models are the attributes of the programs to be

---

[1] *Mean Time Between Failures*

studied. There are many measures of software quality, the most well known being those proposed by [Boehm 78] and [McCall 77]. Both techniques are developed from an intuitive clustering of primitive quality measures. The main difficulty with these approaches is that, although there is agreement on what should be measured, there is little agreement on how best to evaluate and measure them in practice. Finally, even though each model appears to be valid within its own assumptions, there is insufficient experimental evidence available for its large scale validity.

Research most closely related to the present study is in the area of analysis of errors and their causes in large software systems. [Endres 75] discusses and categorizes errors and error frequencies during the internal testing phase of the IBM DOS/VS system. [Hamilton 78] applies the well known execution time model [Musa 80] to measure the operational reliability of computer center software, and [Glass 80] examines the occurrence of persistent bugs and their causes in operational software. Another useful studies is [Maxwell 78], which tabulates and examines error statistics on software.

None of these studies try to relate system reliability or the error frequencies to the usage environment of the software itself in a systematic manner. Results based on such measurements are essential if a scientific basis is to be developed for software reliability evaluation. The argument for adopting a particular approach is more convincing if backed by experiments demonstrating its usefulness.

The operational phase of mature software is somewhat different from the development, debugging, and testing phases. A typical situation is

one where frequent changes and updates are installed either by the installation programmers or by the vendor. Often the vendor will install a change to fix an error found at some other installation, without any notification to the installation management. In a sense the system being measured the represents an aggregate of all such systems maintained by the vendor.

An experimental study therefore provides not only a view of the end product but also gives some insight into the persistent problems. This information can be valuable both in designing new systems and in developing testing strategies for new releases.

In an early study of failures on the SLAC Triplex system[2] [Iyer 82] found a strong correlation between the occurrence of failures (both hardware and software) and the load on the system as measured by variables such as the paging rate and the jobstep processing rate. All failures were considered, not simply the ones which led to system service interruptions. Most importantly the effects were such that the average failure rate for both hardware and software components varied cyclicly over a band of significant width as determined by the daily load variations. Fig. 1 below is a representative histogram from that study of all software failures plotted by the hour of day, averaged over 1978.

A more detailed and accurate analysis on a different system was considered necessary before such results could be considered representative. The VM/370 system on the IBM 3081 at SLAC (in service since February 1981) provided an ideal opportunity in this respect. We commence

[2] At the time of the previous study, the SLAC system consisted of two IBM 370/168s and one IBM 360/91 configured in a triplex mode.

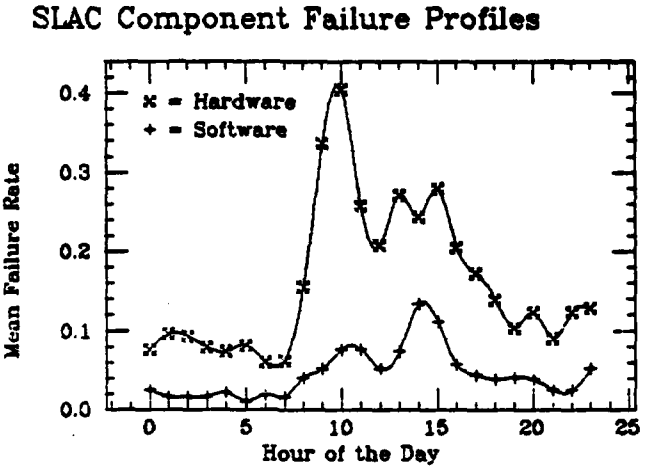


Figure 1: Software failures by hour of day (SLAC Triplex).

by describing our measurements of failures and system activity.

3. SOURCES OF DATA

As explained in the Introduction, we wished to study the occurrence of failures in relation to the system activity at the time of the failure. To begin with, we restricted our analysis to all abnormal terminations of the system. The data on these events came from the system IPL (Initial Program Load) log, automatically recorded by the operating system. Data on system utilization and performance came from the VM accounting and performance system. To avoid the collection of misleading startup data, the first month of normal operation was ignored.

3.1 FAILURE DATA

Failure data for this study originates in an automatically collected log of all IPL's of the system, both scheduled and non-scheduled. For the non-scheduled IPL's the problem is investigated and a determination of the cause of failure is made. This may involve hardware repair or the study of a system dump. Finally, the manager of systems or head system programmer enters the cause of failure. On the basis of their determination the failures were tagged using the following categories:

1. Hardware (HW) - A hardware failure caused the crash.
2. Software (SW) - A software failure caused the crash. If both hardware and software were involved, then the HS category is also indicated.
3. Operator induced (OPR) - Any human error.
4. Unknown (UNK) - Nothing could be blamed.
5. Repeat (RPT) - A reoccurrence of a previous failure.

When software is involved, the following additional categories are defined:

1. CTL - A control or logic problem.
2. ERH - An error handling problem.
3. HSE - A hardware error-handling problem due to lack of robustness in the software.
4. TIM - A timing related problem.

A sample of the online failure data base created on this basis is shown in Fig. 2.

COMMENTS	UPTIME	DOWNTIME	UPHRS	DOWNHRS	MN	SH	CTL	SHH	YSH	OPT	LOK	OPR
TYJ - STC SHUT HANG	17.JUN81:08:46:00	17.JUN81:21:53:00	13.117	0.03333	HH							
IPCS - PGT002 ABEND	18.JUN81:08:46:00	18.JUN81:12:00:00	3.233	0.03333		SH						
TYJ - SHUT HANG UNABLE TO FORCE	18.JUN81:12:02:00	21.JUN81:21:20:00	81.300	0.06667	HH	SH		SHH				
SYSTEM - CRASHED FOR AN UNKNOWN REASON	21.JUN81:21:34:00	22.JUN81:02:09:00	4.250	0.03333		SH	CTL					
REC - PGT002	23.JUN81:00:25:00	23.JUN81:22:13:00	13.800	0.03333		SH	CTL					
TYJ - PGT002	23.JUN81:00:40:00	23.JUN81:17:23:00	8.367	0.03333		SH	CTL			OPT		
TYJ - IBM 3082 PROBLEM	23.JUN81:17:23:00	26.JUN81:10:01:00	16.600	7.26667	HH							
TYJ - IBM 3082 PROBLEM	26.JUN81:17:17:00	26.JUN81:17:31:00	0.233	0.03333	HH							
TYJ - ICC CAUSED SHUH 195 HANG	26.JUN81:19:26:00	29.JUN81:22:06:00	50.000	0.03000	HH	SH		SHH				
TYJ - PROG03 ABEND	29.JUN81:22:09:00	29.JUN81:17:19:00	19.167	0.26667		SH	CTL					
OPERATOR - L COTTRELL POWERED DOWN LOC CONT	30.JUN81:00:51:00	30.JUN81:16:31:00	7.667	0.11667								OPR
OPERATOR - OPERATOR HANG CHANNEL 5 190X	30.JUN81:16:30:00	30.JUN81:17:26:00	0.900	0.10000								OPR
OPERATOR - MEMORY FAILURE - MACHINE CHECK	02.JUL81:09:22:00	04.JUL81:10:44:00	97.367	0.06333	HH							
OPERATOR - SYSTEM CRASH IBM TRAP	07.JUL81:09:08:00	07.JUL81:09:10:00	0.167	0.06667		SH		SHH				
OPERATOR - POWER FAILURE ON 3082	07.JUL81:09:30:00	07.JUL81:17:53:00	8.250	0.20000	HH							
OPERATOR - POWER FAILURE ON 3082	08.JUL81:02:14:00	08.JUL81:02:20:00	0.267	0.03333	HH					OPT		
OPERATOR - PRA003 FAILURE, CP0 190X LOOP	08.JUL81:03:10:00	08.JUL81:12:00:00	8.833	0.06667		SH	CTL	SHH				
OPERATOR - CHANNEL CHECK FROM SERZZ0 1	09.JUL81:09:13:00	09.JUL81:12:42:00	3.483	0.23000	HH							
MCN - VBR003 ABEND	09.JUL81:12:57:00	10.JUL81:17:05:00	28.133	0.08000		SH	CTL					
MCN - IOB009 ABEND	10.JUL81:17:08:00	13.JUL81:13:50:00	68.033	0.15000		SH	CTL					
OPERATOR - PRA002, CP2 LOOP 190X, RE-IPLD	13.JUL81:14:07:00	14.JUL81:12:19:00	22.200	0.41667		SH	CTL					
OPERATOR - PRB005 ABEND.	16.JUL81:08:59:00	16.JUL81:10:23:00	1.400	0.20000		SH	CTL					
OPERATOR - CP0 CP2 190X SUP. STATE..R	16.JUL81:10:35:00	20.JUL81:16:52:00	102.283	0.23333		SH	CTL					
OPERATOR - PRA003 FAILURE, CP0 190X SUPERVI	21.JUL81:08:52:00	21.JUL81:11:19:00	2.450	0.13333		SH	CTL					
OPERATOR - PRA003 CP0 190X SUPERVIS STATE	21.JUL81:11:27:00	21.JUL81:11:44:00	0.283	0.13000		SH	CTL					
OPERATOR - STC TAPE HANG	23.JUL81:08:47:00	24.JUL81:01:56:00	17.183	0.16e37	HH	SH		SHH				

Figure 2: Sample failure data.

3.2 PERFORMANCE AND UTILIZATION DATA

Information on system utilization and performance is provided by the VM accounting and performance system installed and maintained by the SLAC computer center. This data is similar in function to the well known IBM SMF log [SMF 73], although it is generally of superior quality. The data logged consists of the values of a number of workload variables relating to each virtual machine. Statistics are logged every thirty minutes, at logoff, or at disconnect — whichever occurs first. With careful processing these records provide an excellent view of the level and type of system activity. The level is indicated by the absolute values of the workload variables; the type is determined by the nature of usage indicated by the three general categories defined below:

1. Overall Execution

   a) VTIME - Virtual machine processor time (fraction of two pro-

cessors).[3]

b) TTIME - Total processor time (VTIME plus overhead, fraction of two processors).

2. Interactive Execution

a) PAGEIN - Total number of page reads (per second).

b) PAGEOUT - Total number of page writes (per second).

c) SIO (Start I/O) - Total number of non-spooled input/output operations (per second).

3. Others

a) OVERHEAD - Demands placed on the operating system by user jobs (TTIME - VTIME).

b) PRINT - Total number of virtual lines printed (per second).

c) PUNCH - Total number of virtual cards punched (per second).

d) READER - Total number of virtual cards read (per second).

3.3 MATCHING SOFTWARE FAILURES AND SYSTEM ACTIVITY

In order to analyze the level and nature of system utilization in an accurate and efficient manner, a uniform data base containing the values of all the workload variables prior to the occurrence of a failure was created.

The first step was to create 5-minute time averages for all workload parameters for the entire period of our study (March 1981 through April 1982). A sample of this data for PAGEIN and TTIME appears in Fig. 3. For matching purposes, the workload in a specified interval prior to the failure was combined with the failure point. This is illustrated in

[3] The 3081 is a "dyadic" or dual processor. Full utilization is defined to be 2.0.

Fig. 4. After some experimentation the average load in a one hour interval prior to the failure was found to be the most suitable.

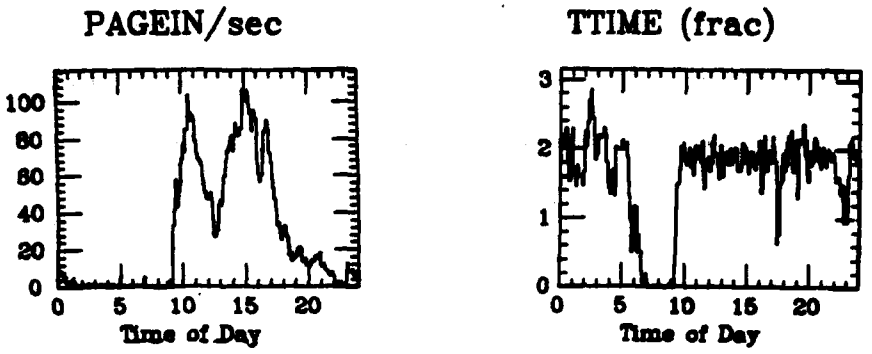


Figure 3: A one day sample: PAGEIN and TTIME

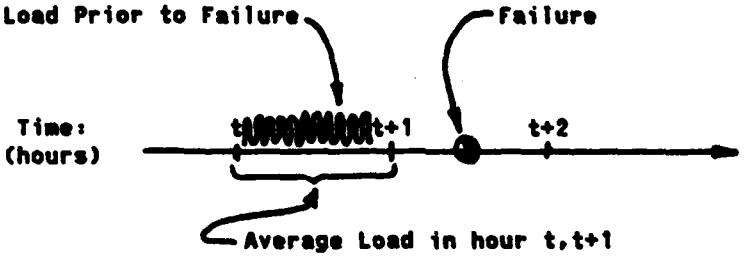


Figure 4: Matching failures and workload.

The creation of these data bases required complex processing in order to minimize the loss of information that invariably accompanies such procedures. The software system developed for this purpose is discussed in [Rossetti 81]. The system is highly interactive and allows efficient handling of large amounts of data of varying formats and complexities.

4. SOFTWARE FAILURE CHARACTERISTICS

We commence our analysis by tabulating some example failure statistics. As a first step we compared our results with those obtained for failures at SLAC on the Triplex configuration [Butner 80]. This comparison is shown in Table 1. It is clear that in terms of MTBF, the new system is at least twice as reliable as the Triplex configuration.

TABLE 1

Mean Time Between Failure Comparison

SLAC Triplex and SLAC 3081 (in hours)

Failure Type	Triplex (1978)	Early 3081 (Mar81-Jun 81)	Late 3081 (Sep81-Apr82)
All	23.19	40.41	69.22
Software	33.10	68.29	110.19
Hardware	90.28	90.79	183.60

Table 2 provides more detailed time-between-failure (TBF) and time-to-repair (TTR) statistics for the 3081 system. The columns correspond to mean, standard deviation, minimum, and maximum values for each measure. The results are also broken down by the major failure categories defined in Section 3.1, and are identified by row. We also divided the time period in our study into two parts: an initial period from March 1981 through August 1981, and a more recent period from September 1981 to April 1982. There are two reasons for this: first, we expected a lower MTBF in the early part of system life than in the later; secondly, the system load began to stabilize and reach peak values much more often during the second part of the study. In interpreting Table 2, note that the sub-categories are not mutually exclusive. A software failure (SW) could also be flagged with ERH if error handling was judged to be part of the problem.

TABLE 2

Failure and repair statistics (SLAC 3081)

TIME BETWEEN FAILURES, TIME TO REPAIR
SLAC VM 3081
Means and Standard Deviations (In Hours)

WHEN	TYPE	N	MTBF	STDTBF	MINTBF	MAXTBF	MTTR	STDTTR	MINTTR	MAXTTR
TOTAL	ALL	177	53.10	76.35	0.13	434.72	0.69	2.00	0.02	19.25
	CTL	63	150.95	191.60	0.22	813.07	0.25	0.73	0.02	5.20
	ERH	46	196.93	268.61	0.55	1136.15	0.62	1.87	0.03	11.25
	HS	20	320.98	449.15	0.62	1548.17	0.22	0.36	0.03	1.98
	HW	72	128.78	163.87	0.18	816.35	1.00	2.61	0.03	19.25
	SW	100	87.48	118.85	0.17	648.35	0.42	1.37	0.02	11.25
EARLY	ALL	99	48.41	49.18	0.13	215.38	0.49	1.25	0.02	8.35
	CTL	32	127.76	119.95	0.22	448.78	0.12	0.11	0.02	0.42
	ERH	25	149.59	245.83	4.35	1136.15	0.25	0.48	0.03	1.98
	HS	14	227.09	438.38	15.08	1548.17	0.25	0.49	0.03	1.98
	HW	43	98.79	128.88	0.18	478.35	0.92	1.88	0.03	8.35
	SW	59	68.29	88.32	0.22	448.78	0.17	0.28	0.02	1.98
LATE	ALL	78	69.22	98.67	0.17	434.72	0.94	2.66	0.02	19.25
	CTL	31	174.13	263.63	1.07	813.07	0.38	1.03	0.02	5.20
	ERH	23	246.32	271.72	0.58	976.28	1.02	2.64	0.05	11.25
	HS	14	408.16	464.29	0.62	1324.65	0.28	0.15	0.08	0.62
	HW	29	183.68	200.22	0.58	816.35	1.13	3.51	0.05	19.25
	SW	49	118.19	150.21	0.17	648.35	0.71	1.98	0.02	11.25

In the TBF columns, it is interesting to see the dramatic improvement in reliability between the early and late data. For example, hardware reliability more than doubled and software reliability improved by almost 60 percent. From the TTR columns we can see that, as expected, hardware failures cause longer down times (avg. 1 hour) than software failures (avg. 0.42 hours). The table also shows that the longest down period was 19.25 hours, and that it was due to a hardware problem.

Table 3 below gives all the unique failure patterns, with their frequency of occurrence. We notice that failures where only hardware was involved account for just 20% of all system reloads, while over 60% of all cases relate to a software problem. That 60% includes situations where both hardware and software were involved (16% of the total). In most of the hardware/software cases (approximately 12% of all failures) a common scenario was that a hardware failure made the system go into a region of the software which was not sufficiently robust to handle the problem (a "hardware/software error handling problem"). Of the remaining software failures, control problems (35% of all failures) and error handling problems dominate (27% of all failures). Synchronization and timing window problems account for 10%. We also notice a rather large share of repeats - approximately 15%.

TABLE 3

Software failure patterns

MM	SM	CTL	ERR	UNK	OPR	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
					OPR	17	17	9.605	9.605
				UNK		8	25	4.520	14.124
	SM					3	28	1.695	15.819
	SM		ERR			18	46	10.169	25.989
	SM	CTL				54	100	30.508	56.497
	SM	CTL	ERR			5	105	2.825	59.322
MM						43	148	24.294	83.616
MM			ERR			1	149	0.565	84.181
MM	SM		ERR			24	173	13.559	97.740
MM	SM	CTL				4	177	2.260	100.000

4.1 STATISTICAL TESTS

In common with other analyses of this type, our first test was to investigate the distribution of the time between failures. A Kolmogorov-Smirnov test conducted on the time between failure distribution accepted at any level of significance a Weibull distribution with the following density function and parameters:

$$f(t) = \alpha \beta t^{\beta-1} \exp [-\alpha t^{\beta}]$$

where: $\alpha = 0.092$ (characteristic life of 40.8 hours)

$\beta = 0.647$

Since $\beta < 1.0$, this is clearly a distribution with a decreasing failure rate in time. The empirical and Weibull cumulative distribution functions are plotted in Fig. 5. This also conforms with the plot of the monthly average failure rates in Fig. 6.

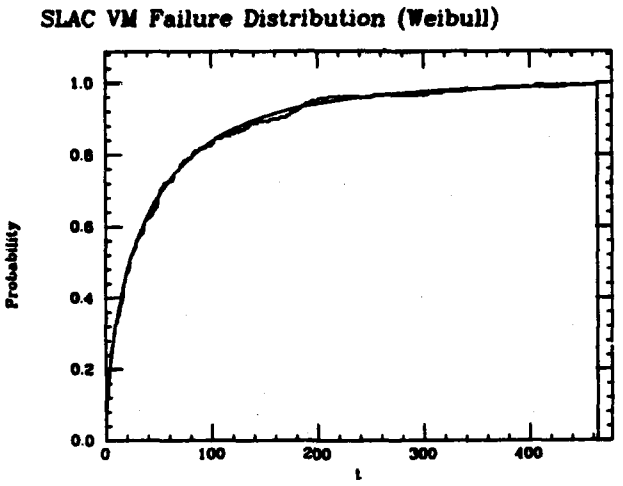


Figure 5: Theoretical and empirical Weibull distributions (cdf)

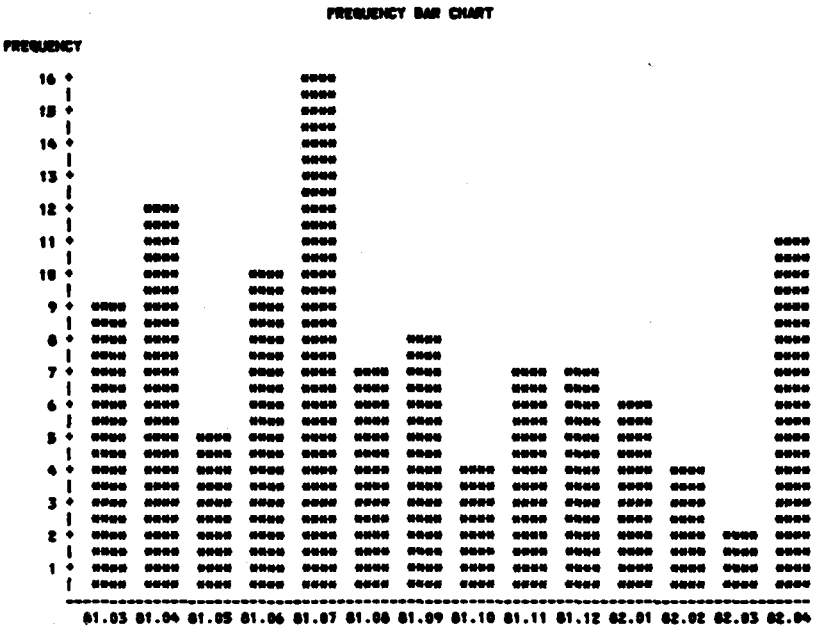


Figure 6: Average software failure rate (by month)

5. SOFTWARE FAILURES - DEPENDENCE ON SYSTEM ACTIVITY

In this section we attempt to relate the occurrence of software failures to the type of system utilization as measured by the various workload variables. It is envisioned that such experiments would provide insight into the most probable "cause and effect" relationships for software failures.

It is to be expected that most workload measures will be cyclic on a daily basis. Accordingly, it was instructive to examine the mean software failure rate behavior over the same period [Beaudry 78]. This pro-

vided not only a quick visualization of significant failure trends but was also useful in developing subsequent statistical experiments.

Figure 7 shows a histogram of failures by hour of day for all software related failures. Note the sharp rise in failure rate during early morning hours. This appears to follow rather closely the typical interactive load at SLAC and also compares favorably with a similar plot of software failures for the SLAC Triplex configuration (Fig. 1).

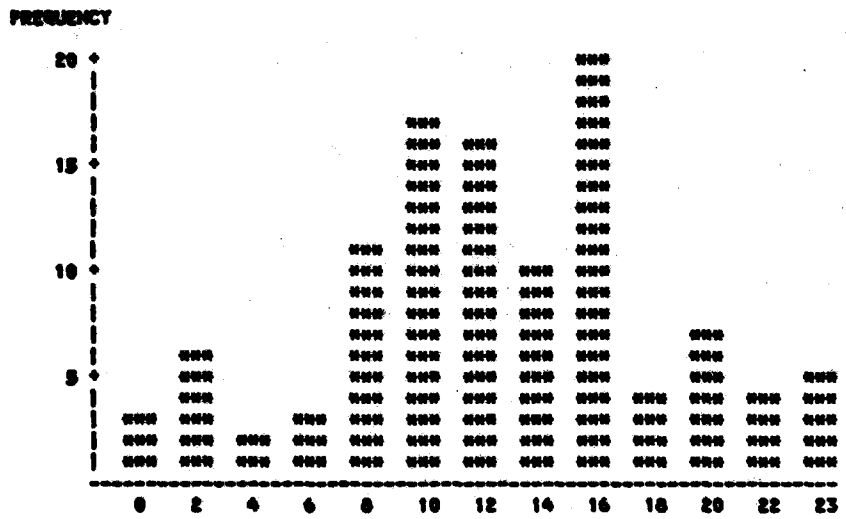


Figure 7: Software failures by hour of day (SLAC 3081)

5.1 DISTRIBUTIONS OF FAILURES AND SYSTEM ACTIVITY

As explained in Section 3.1, our data provided us with a set of system workload measures. In particular, measures such as the paging (PAGEIN and PAGEOUT) and input/output (SIO) rates provide a measure of the system interactive load, while measures such as TTIME and VTIME provide a general view of the CPU usage. The variable "OVERHEAD", derived from the difference between TTIME and VTIME, is a direct measure of the demands being placed on the operating system by users' programs (actually virtual machines).

Recall that the data base developed contains not only the values for the specified workload variables to a five minute resolution but also the values of the same variables matched with failure times. From this data three types of distributions were generated. The first, $\mathcal{L}(x)$, is simply the distribution of the workload in question.

$$\mathcal{L}(x) = \Pr\{\text{Workload Measure} = x\}$$

The second is the joint distribution of the failure and the workload measure:

$$f(x) = \Pr\{\text{Failure Occurs and Workload} = x\} .$$

This is easily obtained from the failure matched data base, the generation of which was described in Section 3.1.

In f(x) both the failures and the workload measures are represented as they occur in the system. Clearly the more favored values for a given workload will contribute more to this distribution than the less favored

ones.[9] Using the well known notion of conditional probability, we define:

$$g(x) = Pr\{\text{Failure Occurs} \mid \text{Workload} = x\} = \frac{f(x)}{\ell(x)}.$$

g(x) can now be thought of as the probability of a software failure at a given value for workload when all values are equally represented. Figure 8 shows the plots for $\ell(x)$, f(x), and g(x) for three selected workload variables: PAGEOUT, SIO, and TTIME; All software failures are considered; see the appendix for other variables and subclasses.

As a general observation we note that, where the difference between $\ell(x)$ and f(x) is considerable, we might expect to see a workload dependency in the failures. If $\ell(x)$ and f(x) are similar, the relationship is probably not significant. A g(x) distribution sharply weighted in favor of higher workload values will clearly generate a higher risk of failure as the load increases.

It would appear from the g(x) plots for PAGEOUT and SIO that higher values of these measures (> 10 for PAGEOUT) contribute more significantly to software related failures than the lower values. Examining the plots for TTIME we note that, as measured by CPU utilization, the system was heavily loaded (close to 2.0) most of the time. The $\ell(x)$ and g(x) plots for TTIME show considerable similarity. It would therefore appear from this cursory analysis that failures are not induced by higher execution rates, as measured by CPU usage.

---

[9] A rather commonplace analogy to illustrate this is that automobiles travelling at 150 mph have a higher probability of an accident than those travelling at 55 mph. There are however far fewer accidents at 150 mph. To obtain an accurate representation of the risks involved in travelling we must divide the number of accidents at high speeds by the number of autos travelling at that speed.

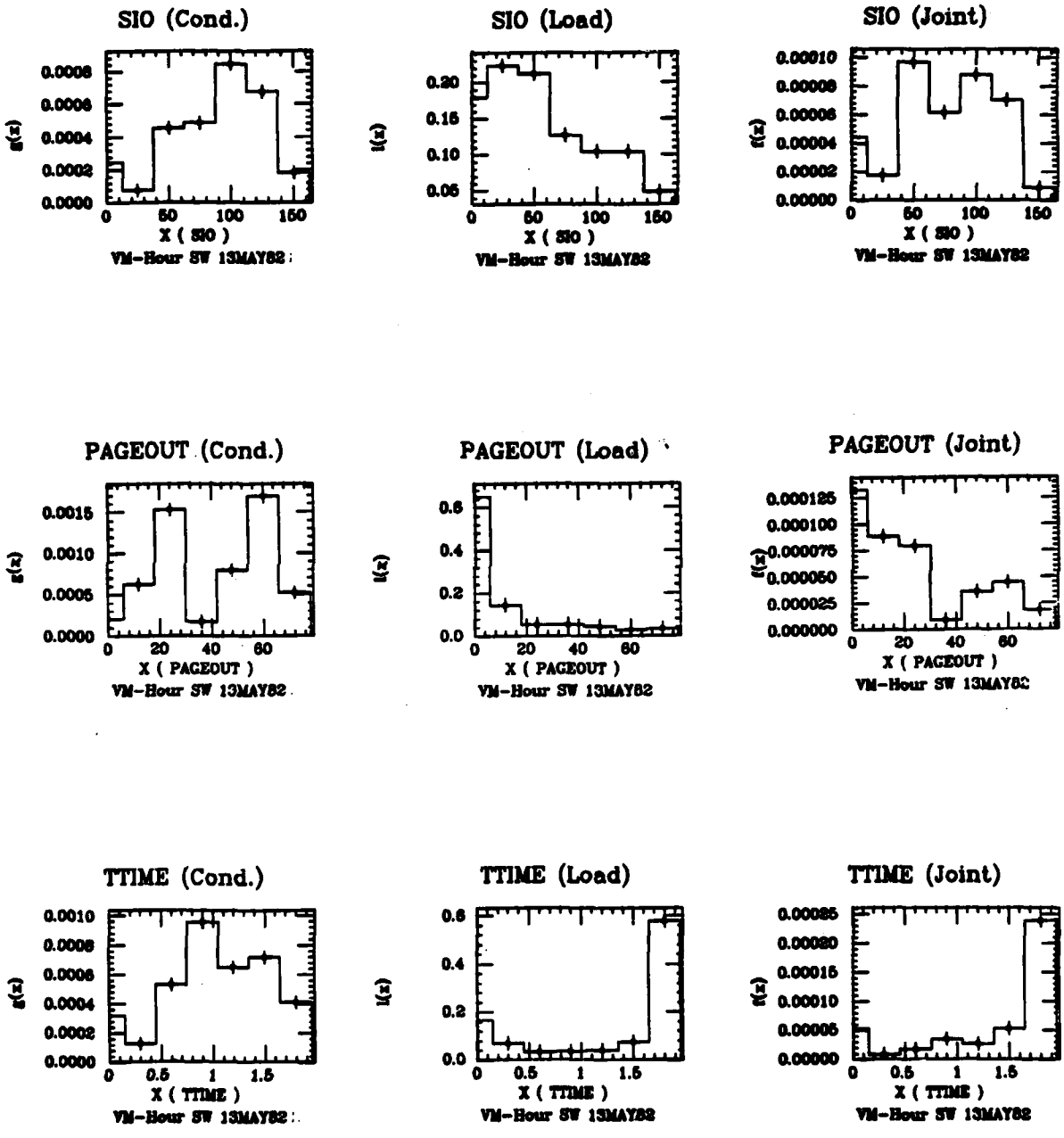


Figure 8: Frequency distributions: $\ell(x)$, $f(x)$, and $g(x)$

In order to quantify this effect, in particular to determine exactly the risk or "hazard" associated with higher workload values, we employed what we refer to as a "load hazard" model, the development and application of which is discussed in the next section.

5.2 A SOFTWARE LOAD HAZARD MODEL

The object of our analysis was to determine:

1. Does a higher level of system utilization result in a higher risk of failure than a lower level?
2. Is the relationship linear with the workload variables, or is there a nonlinear increasing effect?

In practical terms, if such an effect exists, we expect the load to act as a stress factor. For this purpose we developed and validated a load-hazard model which formed the basis for our tests. A detailed description of the development and validation of this model is discussed in [Iyer 82]. Briefly, an inherent load hazard z(x) is defined as

$$z(x) = \frac{\Pr\{\text{Failure in load interval } (x, x+\Delta x)\}}{\Pr\{\text{No failure in load interval } (0, x)\}} = \frac{g(x)}{1 - G(x)} \qquad (1)$$

where:

g(x) is as defined in section 5.1, and

G(x) is the cumulative distribution function of g(x).

In close analogy with with the classical hazard rate in reliability theory [Shooman 68], z(x) measures the incremental risk involved in increasing the workload from x to x+Δx. If z(x) increases with x, it should be clear that the there is an increasing risk of a failure as the workload variable increases. If, however, z(x) remains constant for increasing x, we may surmise that no increased risk is involved.

Note that in our definition of load hazard we have removed the variability of system load by using the conditional probability g(x). This of course is not true in practice since load is best described as a random variable with a probability distribution; it is simply the associated load distribution, $\ell(x)$, defined above. In order to determine the hazard for a particular load pattern, we must multiply the associated load probability by the hazard calculated in (1). Denoting by $z_a(x)$ the transformed hazard, we have

$$z_a(x) = z(x)\,\ell(x) \qquad (2)$$

We refer to the hazard z(x), as defined in (1), as the <u>fundamental</u> hazard. This is because it can be thought of as an inherent property of a particular system and is not subject to varying load patterns. When a varying load pattern is taken into account, it can be thought of as "picking out" aspects of the fundamental hazard function. This hazard $z_a(x)$ defined in (2) will be referred to as the <u>apparent</u> hazard, since it is closely dependent on the load distribution.

The following example illustrates how a particular workload can modify a given fundamental load hazard z(x). Figure 9(a) shows a sample fundamental hazard z(x). Note that z(x) is increasing with load. Thus, if all load values are equally likely, the system has a higher risk of failure at higher load values than at lower load values. Figure 9(b) is a hypothetical load distribution where the load variable is the fractional CPU utilization, with 0 for an idle CPU and 1 for a fully busy CPU. Finally, Fig. 9(c) gives the apparent hazard due to the effect of the load distribution in (a). The apparent hazard is now decreasing simply because higher load values are less probable.

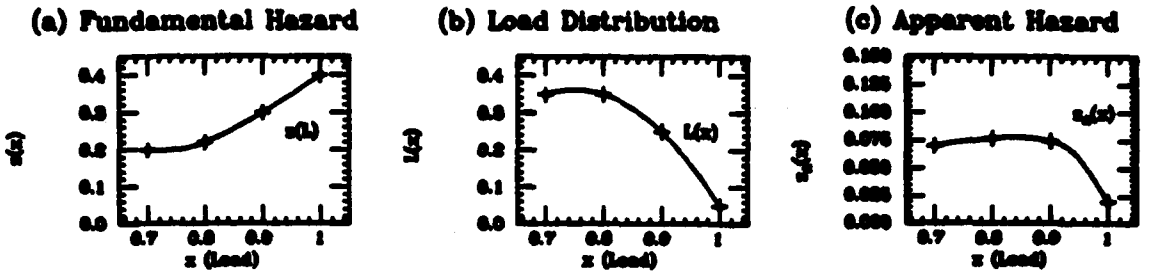


Figure 9: Example of fundamental and apparent hazards

5.3 HAZARD PLOTS

The generation of the hazard plots and associated statistics involved extensive data processing. In each hazard plot, z(x) or $z_a(x)$ is calculated and plotted as a function of a chosen workload variable, x. In developing hazard plots for the load-failure data, those factors not related to load are expected to behave as noise in a load-failure analysis. If such factors are predominant, we can expect to find no discernable pattern in our hazard plots, i.e. they should appear as uncorrelated clouds.

An easily discernable pattern, on the other hand, would indicate that the load-failure dependency dominates others. The strength of such a relationship can be measured through regression. Figs. 10, 11, and 12 depict the hazard plots for three selected load parameters (PAGEIN, SIO, and OVERHEAD). These plots relate to all software failures; see the appendix for other variables and subclasses. The regression coefficient R^2, which is an effective measure of the goodness of fit, is provided for each plot. Quite simply, it measures the amount of

variability in the data that can be accounted for by the regression model. R^2 values of greater than 0.6 (corresponding to an $R > 0.75$) are generally interpreted as strong relationships [Younger 79].[5]

It would appear from our data that many of the workload parameters are acting as a stress factor, i.e., that there is an increasing risk of failure with increasing load. In the case of the interactive workload measures OVERHEAD and SIO there is no doubt that, statistically, there is an increased risk of a software failure as the load increases. The correlation coefficients of 0.95 and 0.91 show that a very close fit was obtained and that the failures closely fit an increasing load-hazard model. The risk of a failure also appears to increase with increased PAGEIN, although at a somewhat lower correlation ($R = 0.82$). Importantly, we note that:

1. We are not seeing a statistically higher failure rate simply due to greater execution. With CPU usage (TTIME) as a measure, one finds that the correlation is unacceptable, i.e., that no relationship exists. This would appear further proof that simply a greater execution rate (as measured by CPU utilization) is not a major cause of the observed failures.
2. The relationship is highly non-linear, i.e, the risk of a failure markedly increases as workload variables reach peak values. This tends to indicate that there is a complex set of interactions that adversly affects the operating system as end points are reached.

---

[5] The range of $|R|$ from 0 to 1 is typically divided as follows: (0, 0.25) moderately weak; (0.25, 0.5) moderate; (0.5, 0.75) moderately strong; (0.75, 1.0) strong.

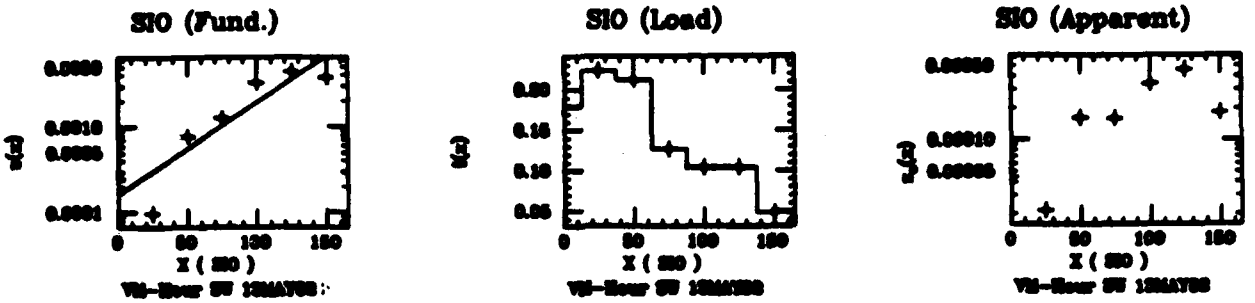


Figure 10: Hazard plots: SIO

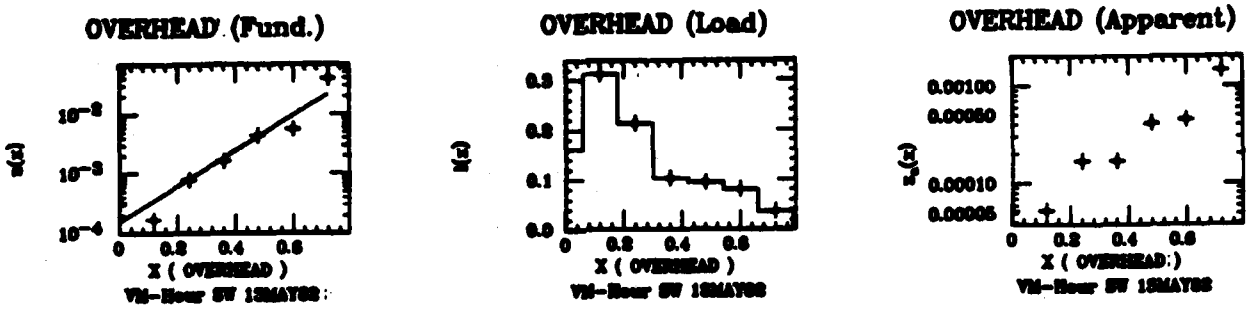


Figure 11: Hazard plots: PAGEIN

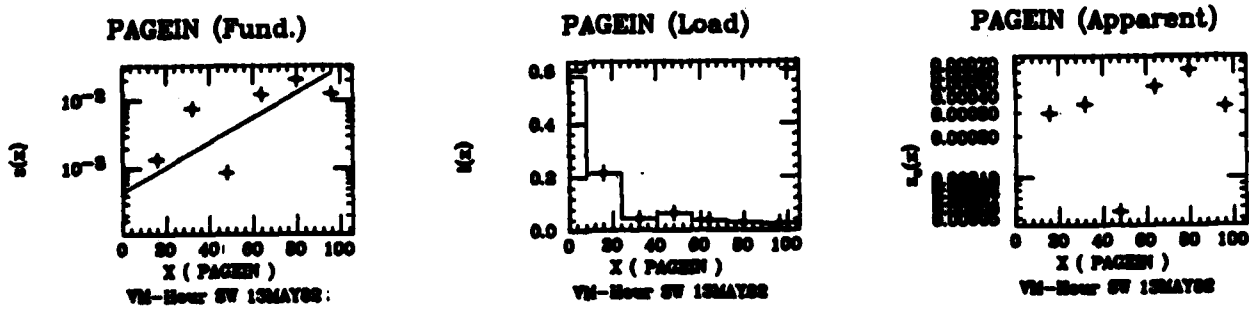


Figure 12: Hazard plots: OVERHEAD

The vertical scale is logarithmic in these plots, indicating that the hazard is rising sharply at peak loads.

In the next section we provide conjectures on the possible causes of this dependency and provide further interpretation of our results.

6. VIEWS ON OBSERVED RESULTS

The question of fault tolerant design has been studied by a number of authors (see for example [Hecht 80a 80b] and [Yau 80]). In order to be failure tolerant, the software must be able to deal with adverse effects in a well defined manner. The ideal situation is one where an error is detected at the earliest time, thus containing the impact of the error to its minimum.

It is clear from our analysis that, in practice, we are far from this ideal, even in a well structured system such as VM/370. What we observe is that often the most severe malfunctions occur when the workload becomes more complex. Due to the extensive degree of inter-user and user-system dependency under these circumstances, it is usually not possible to contain the impact of the error and a system crash ensues.

Each of the following sections discusses a particular way that system failures are thought to relate to the quality and quantity of workload. Examples of typical SLAC failures are given in each case. Note that a detailed analysis was performed on every system crash to determine the exact cause of failure. This was done by careful tracking and record keeping by the SLAC system support staff.

6.1 SYSTEM DESIGN ASSUMPTIONS

At numerous points in the design and coding of system components implicit and explicit assumptions are made about the environment the component will be subjected to. The three cases below characterize the most popular types of assumption-related failures.

Queue, Buffer, and Table Limits: Usually these are found only during extreme situations, where load is abnormally high or unusual.

Example: Recently the number of users logged onto the SLAC VM/370 system went above 250. A system component failed when its table could hold only 250 entries. The result was not catastrophic, but it did affect system monitoring.

Synchronization Assumptions: These assumptions are not usually explicit. They are most often due to the programmer or designer not being able to consider that a certain sequence of events could occur as slowly or quickly as might happen under extreme conditions.

Example: In a recent case a user waited a long time between typing his user id and his password while logging on. During that period the user directory (containing password information) was updated by other users (changing passwords, etc.). When the password was finally entered, the system crashed because the logon process was using outdated pointers into the directory data structure.

Unanticipated State Changes: Many of the bugs discovered relate to an operation that is somehow preempted by an external event. Typically a critical section was not adequately protected from the event and a data structure or program was forced into an inconsistent state.

Example: A number of these bugs have involved a sudden log off during an operation being performed on a user's virtual machine. The operation is only partly completed and a system data structure is made unusable.

6.2 ERROR HANDLING FAILURES

This is an important error category, comprising roughly 27% of all system failures. A claim made for the VM/370 type of operating system structure is that, because of the isolation of users and system functions, reliability can be much better [Donovan 76]. One rationale for a hierarchical system is that the "vertical" segregation of system functions into a hierarchy and the "horizontal" isolation of users from each other affords easier fault isolation and recovery. Concievably, offending users and components can be removed in many cases without loss of the system. We agree, and feel that the relatively high reliability of the SLAC system is due in part to the VM/370 design. However, some of the resiliency expected is not implemented, causing error handling to be involved in a large fraction of system crashes. We divide this category as follows:

6.2.1 Hardware Induced Errors

About 22% of software failures (24 of 108) involved the failure of the software to continue after a non-catastrophic hardware error. These only include cases where it was determined that the system should have been designed to continue, possibly but not necessarily, in a degraded mode.

Previous studies show that system activity, especially I/O activity, is strongly related to processor failure rates [Iyer 82] [Castille 81]. Since operating systems are required to react to such failures it can be expected that more software failures will occur.

Input/Output Errors: It is clear that I/O errors are directly related to the I/O rate or the amount of data being transferred. Since a nontrivial fraction of all I/O can critically affect system operation, the exposure to system failure can be expected to increase.

Example: Errors while transferring memory pages from or to an external device can be catastrophic. Unfortunately, many systems do not adopt a strategy of graceful degradation in such an area; the next section addresses this question.

Microcode Errors: Most modern computer systems rely heavily on microcode in various system components. In the IBM 3081 its use is pervasive - controlling essentially all hardware components from the operator's console to performing failure diagnosis [Reilly 82]. Microcode even controls and monitors the 3081's power supplies and thermal state in real time. Such a complex microcode system provides a variety of rare states to be entered during intricate event sequences.

Example: The 3082 processor controller for the 3081 has been responsible for a number of failures due to both microcode and hardware failures.

6.2.2 Non Hardware Induced Errors

The results given earlier in this paper demonstrate that software failures will occur more under high system stress. It therefore follows that greater robustness is needed in handling error situations. About 21% of software failures (23 of 108) involved the detection of inconsistencies in system data structures or a weak response to the failure of a particular software component. In almost all of these cases it seemed that the control program should have been able to sever or mend the ailing component and either recover or degrade gracefully. In these failures blame could not be placed on the harduare.

6.3 SOFTWARE CONTROL ERRORS

This category corresponds generally to the classical meaning of a bug. There are two levels of behavior in this category. The first involves the discovery of latent errors; the second relates to the violation of space or timing constraints.

6.3.1 Discovery of Latent Errors

A process inherent in the life of a mature production system is the discovery of latent (or dormant) bugs [Musa 80]. The relation of this type of error to workload is evident. Well used sections of code tend to be more reliable simply because bugs have already been discovered and removed. Under normal loads these sections tend to be heavily used and the system remains reliable. However, during periods of stress or uncommon workload patterns, rarely used code can be executed, leading to the discovery of errors.

Example: A prime example of this phenomenon involved a section of system initialization code to handle the case of finding a faulty storage page frame. Since, in the period of over a year, the 3081 had not encountered such an error, the code had never been executed. The first time a defective frame was found, an obvious coding error was uncovered. In this case the system could not even be restarted to repair the error.

6.3.2 Space and Time Violations

In a typical timesharing environment the variety of demands made on the system (complexity of the load) is directly related to the number of users on the system. Although it is not necessarily true that the number of program states will increase with load, it is clear that the number of timing and data structure states will. We observe that this increase is greater than linear with the workload variables presented in this paper. In fact, this mushrooming of states may explain the exponential increase of failure hazard with load.

One practical way to study control failures is to classify the errors into violations in space (e.g., overwriting storage, invalid operations) and violations in time (e.g., simultaneous update, invalid sequence of operations, insufficient locking of critical data). Experience has shown that in a stable system such as VM/370 the space violations are culled more quickly than timing violations because they:

1. are easier to understand;
2. usually manifest their effects immediately and disastrously;
3. tend to be unaffected by the dilation and contraction of time scales caused by load.

On the other hand, timing related problems can linger in a system for years and can be particularly sensitive to load variations. These errors are more difficult to diagnose because specific load patterns may be required to reproduce the problem and because the manifestation of timing bugs is usually subtle and complex.

Example: At times a failure will occur that is a combination of both a time and a space violation. Typically a complex set of events will lead to a timing error, which triggers the overwriting of an area of storage. Such bugs are extremely difficult to diagnose.

6.4 GRADUAL DECAY OF THE SYSTEM

A new class of non-catastrophic errors begins to surface as a system becomes more reliable. These have to do with the gradual loss of system resources, such as memory frames or free disk blocks due to rare housekeeping errors. Since, typically, these resources are redefined at each system reload, in a relatively unreliable system their loss may never be noticed by the system or its users. If, on the other hand, the system runs for weeks without failure, then the gradual loss can become noticeable.

Example: In the SLAC system, an unknown bug had existed for years that allowed temporary disk space to be lost in small increments over a period of time. After a 10 day period without a system reload, users began to complain about the lack of scratch disk space. Investigation showed that the sum of allocated and free space did not sum to the amount "known" to be available, and the error was corrected.

7. CONCLUSION

It has been the purpose of this paper to present an analysis of software related system failures on the IBM 3081 at SLAC. We find three broad categories of failures: error handling, program control or logic, and hardware-related. A statistical analysis of these failure modes shows (not unexpectedly) a decreasing failure rate with time. This is especially true in the early part of the study. Not withstanding the decreasing failure rate with time, we find that the occurrence of the failures is strongly correlated with the type and level of workload prior to the occurrence of a failure. For example, it is shown that the risk of a software related failure increases in a non-linear fashion with amount of interactive processing, as measured by parameters such as the paging rate and the amount of overhead. The overall CPU execution rate, though measured to be close to 100% most of the time, is not found to correlate with the occurrence of failures. We propose a load-hazard model to statistically measure the above effects. Finally the paper offers conjectures on the observed phenomenon.

As with any statistical analysis, this is not proof in itself. However, the increasing body of evidence accumulated on different computers with differing load and failure patterns shows that workload should be considered as a factor in reliability. The design of computer systems will be greatly aided if this type of analysis can help uncover cause and effect relationships in software failures.

8. ACKNOWLEDGMENTS

The authors would like to thank Prof. E. J. McCluskey for his overall guidance. Special thanks are extended to Ted Johnston and Bill Weeks at SLAC for providing valuable insight into the reliability aspects of the SLAC system. We also are grateful to Dr. David Lu and Dr. George Rossmann for their careful reading and comments on a draft of this paper. This work was supported in part by the U. S. Army Research Office under contract number DAAG29-82-K-0105, and by the Department of Energy under contract number DE-AC03-76F00515.

The views, opinions, and/or findings contained in this document are those of the author and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

REFERENCES

[Beaudry 79] M. D. Beaudry, "A statistical analysis of failures in the SLAC Computing Center," Digest of Papers, COMPCON Spring 79, pp 49-52, 1979.

[Butner 80] S. E. Butner and R. K. Iyer, "A statistical study of reliability and system load at SLAC", Digest, Tenth International Symposium on Fault Tolerant Computing, October 1980.

[Boehm 78] B.W. Boehm, J.R. Brown, H. Kasper, M. Lipow, G.J. McCleod, and M.J. Merrit, Characteristics of Software Quality, Amsterdam, The Netherlands : North Holland, 1978.

[Curtis 80] B. Curtis, "Measurement and experimentation in software engineering", Proceedings of the IEEE, vol. 68, no. 9, September 1980, pp. 1144-1157.

[Denning 80] P.J. Denning, "On learning how to predict", Proceedings of the IEEE, vol. 68, no. 9, September 1980, pp. 1099-1103.

[Donovan 76] J. J. Donovan and S. E. Madnick, "Virtual machine advantages in security, integrity, and decision support systems," IBM Systems Journal, 15, No. 3, pp. 270-278, 1976.

[Endres 75] A. Endres, "An analysis of errors and their causes in systems programs", IEEE Trans. Software Engineering, vol. SE-1 no. 2 June 1975, pp. 140-149.

[Glass 80] R.L. Glass, "Persistent software errors", IEEE Trans. Software Engineering, vol. SE-7, no. 2, March 1981, pp. 162-168.

[Goel 80] A.K. Goel, "A summary of the discussion on 'An analysis of competing software reliability models'", IEEE Trans. Software Engineering, vol. SE-6, September 1980, pp. 501-502.

[Hamilton 78] P.A. Hamilton and J.D. Musa, "Measuring reliability of computation center software", Proc. Third Int. Conf. Software Engineering, Atlanta GA, May 10-12 1978, pp. 29-36.

[IBM 73] IBM Corp., OS/VS System Management Facilities (SMF), Order No. GC35-0004, 1973.

[Hecht 80a] H. Hecht, "Current issues in fault tolerent software", Proceedings COMPSAC 80, 1980, pp. 603-607.

[Hecht 80b] H. Hecht, "Mini-tutorial on software reliability", Proceedings COMPSAC 80, 1980, pp. 383-385.

[Iyer 81] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship: Results of a multi-computer study," to appear in the IEEE Transactions on Computers, July 1982.

[Iyer 82] R. K. Iyer and D. J. Rossetti, "A statistical load dependency model for CPU errors at SLAC," to appear in The Dig. FTCS-12, Twelveth International Symposium on Fault Tolerant Computing, June 1982, Santa Monica Calif.

[Littlewood 80] B. Littlewood, "Theories of software reliability: How good are they and how can they be improved?", IEEE Trans. Software Engineering, vol. SE-6, no. 5, September 1980, pp. 489-500.

[Maxwell 78] F.D. Maxwell, The determination of measures of software reliability, Final Report, NASA-CR-158960, The Aerospace Corporation, El Segundo CA, December 1978.

[McCall 77] J.A. McCall, P.K. Richards, and G.F. Walters, "Factors in software quality", Tech. Report (77CIS02), Sunnyvale, CA, General Electric, Command and information systems, 1977.

[Milliar-Smith 81] P. M. Milliar-Smith and R. L. Schwartz, "Current progress on the proof of SIFT," The Dig. FTCS-11, Eleventh International Symposium on Fault Tolerant Computing, Portland, Maine, June 1981.

[Musa 80] J. Musa, "The Measurement and management of software reliability", Proc. IEEE, vol. 68, September 1980, pp. 1131-1143.

[Reilly 82] J. Reilly, A. Sutton, R. Nasser, and R. Griscom, "Processor Controller for the IBM 3081," IBM J. of Research and Development, 26, No. 1, Jan. 1982, pp. 22-29.

[Rossetti 81] D. J. Rossetti and R. K. Iyer, "A Software System for Reliability and Workload Analysis", CRC Tech. Rpt 81-18, Center for Reliable Computing, Computer Systems Laboratory, Stanford Univ., Stanford, CA. December, 1981.

[Shooman 68] M. L. Shooman, Probabilistic Reliability: An Engineering Approach, McGraw Hill, 1968.

[Wensley 78] J. Wensley, et. al., "SIFT: Design and analysis of a fault tolerant computer for aircraft control," Proc. IEEE, Vol. 66, No. 10, October 1978, pp. 1240-1254.

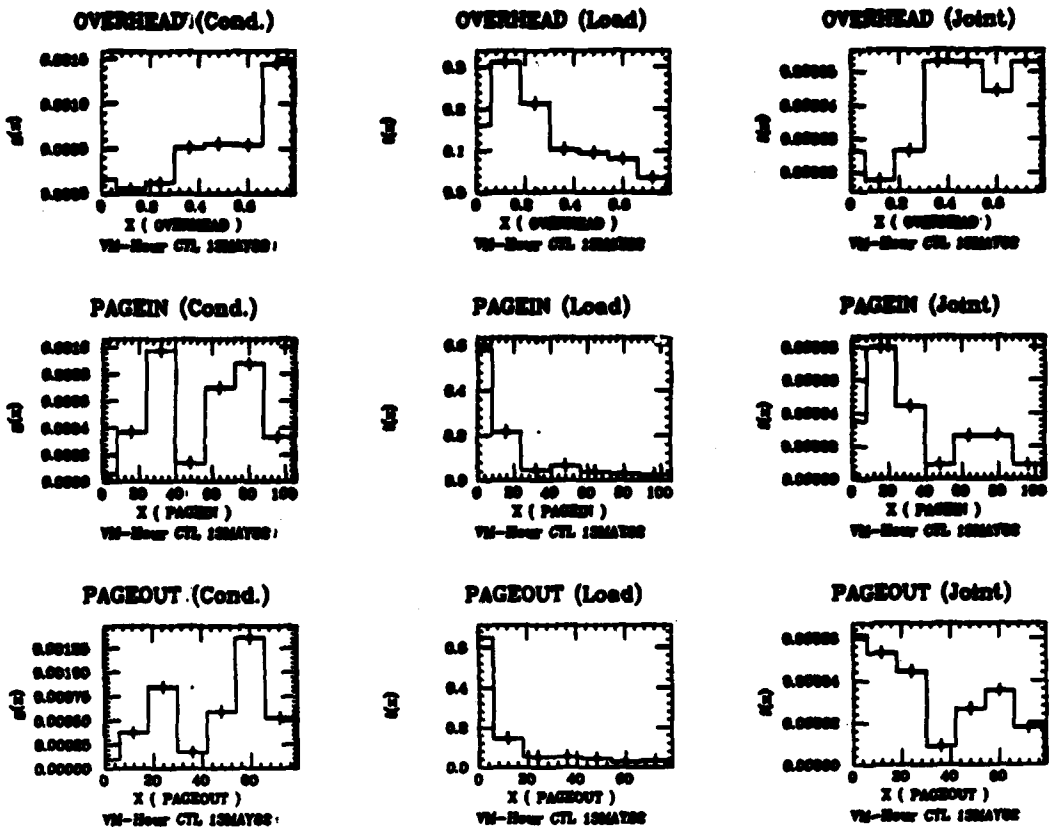
[Yau 80] S.S. Yau, R.C. Cheung, and D. C. Cochrane, "An approach to error-resistant software design", Proceedings COMPSAC 80, 1980, pp. 429-436.

[Younger 79] M. S. Younger, A Handbook for Linear Regression, Wadsworth Inc., 1979.

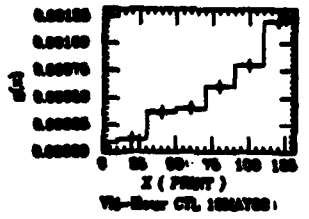
APPENDIX

Load, Failure, and Hazard Plots
For three error types:
Control (CTL), Error Handling (ERH), Software (SW)

The top half of each page contains the Conditional Failure, Load, and Joint Failure distributions. The bottom half shows the corresponding Fundamental and Apparent Hazard functions. The error type is indicated in small type just below each plot; they are arranged in groups of three pages each.


OVERHEAD (Cond.)
OVERHEAD (Load)
OVERHEAD (Joint)
PAGEIN (Cond.)
PAGEIN (Load)
PAGEIN (Joint)
PAGEOUT (Cond.)
PAGEOUT (Load)
PAGEOUT (Joint)
OVERHEAD (Fund.)
OVERHEAD (Load)
OVERHEAD (Apparent)
PAGEIN (Fund.)
PAGEIN (Load)
PAGEIN (Apparent)
PAGEOUT (Fund.)
PAGEOUT (Load)
PAGEOUT (Apparent)

PRINT (Cond.)



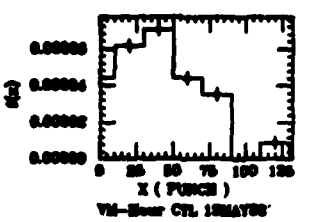
PRINT (Load)

PRINT (Joint)

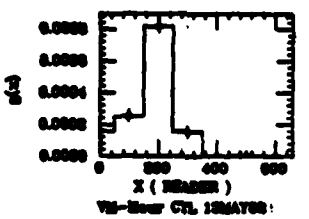
PUNCH (Cond.)

PUNCH (Load)

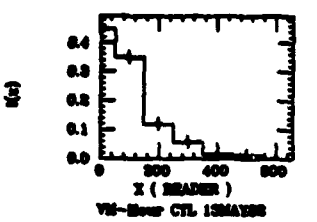
PUNCH (Joint)



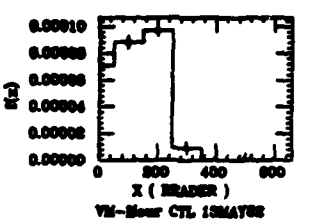
READER (Cond.)



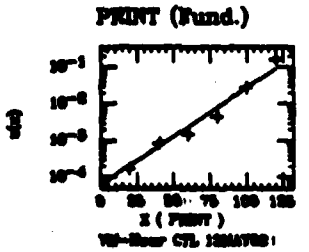
READER (Load)



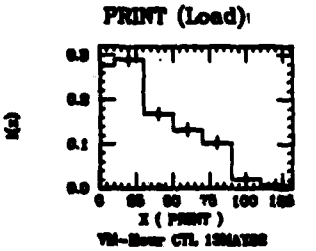
READER (Joint)



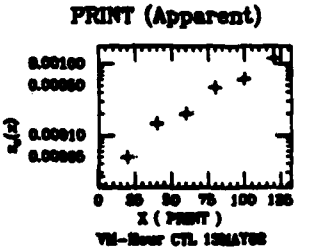
PRINT (Fund.)



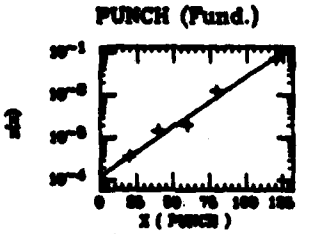
PRINT (Load)



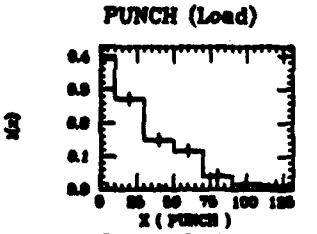
PRINT (Apparent)



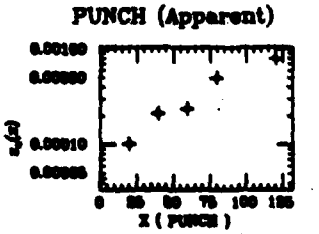
PUNCH (Fund.)



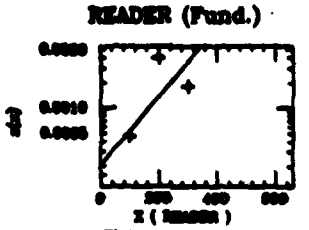
PUNCH (Load)



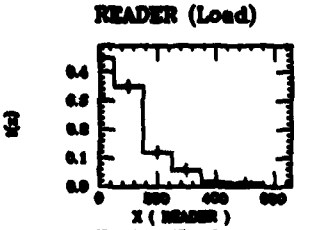
PUNCH (Apparent)



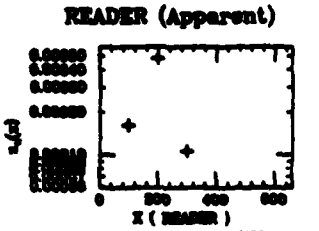
READER (Fund.)



READER (Load)



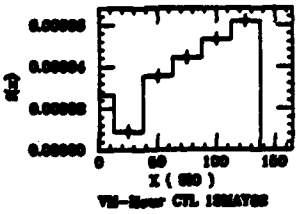
READER (Apparent)



SIO (Cond.)

SIO (Load)

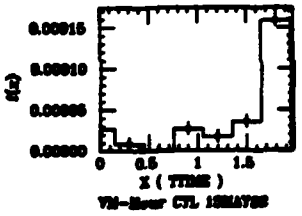
SIO (Joint)



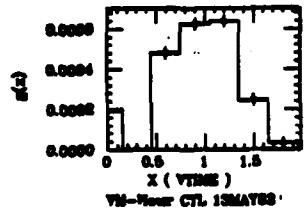
TTIME (Cond.)

TTIME (Load)

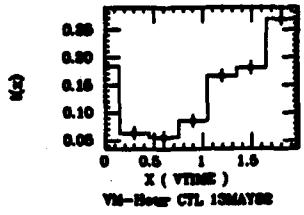
TTIME (Joint)



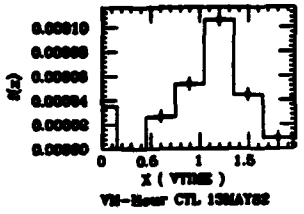
VTIME (Cond.)



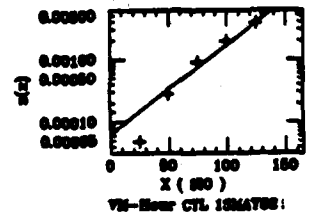
VTIME (Load)



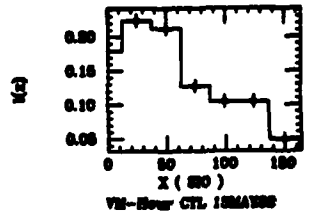
VTIME (Joint)



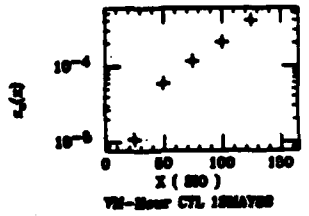
SIO (Fund.)



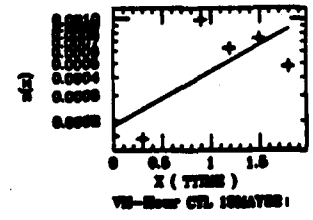
SIO (Load)



SIO (Apparent)



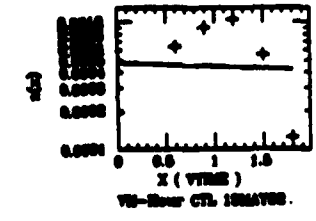
TTIME (Fund.)



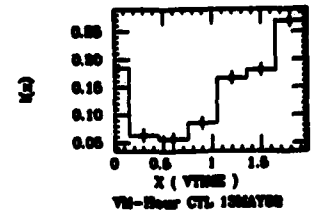
TTIME (Load)

TTIME (Apparent)

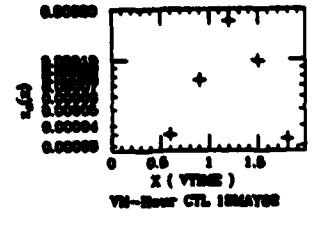
VTIME (Fund.)



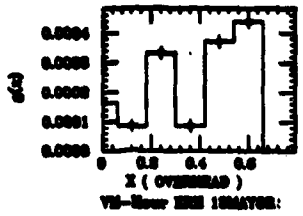
VTIME (Load)



VTIME (Apparent)

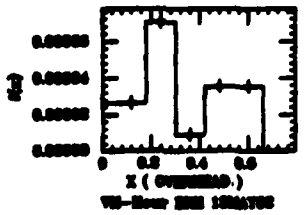


OVERHEAD (Cond.)



OVERHEAD (Load)

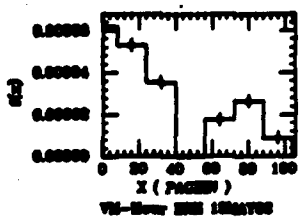
OVERHEAD (Joint)



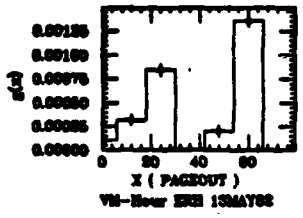
PAGEIN (Cond.)

PAGEIN (Load)

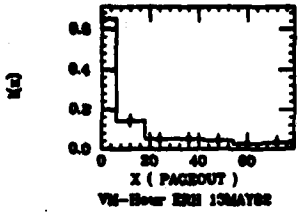
PAGEIN (Joint)



PAGEOUT (Cond.)

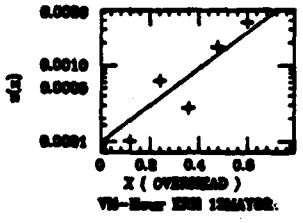


PAGEOUT (Load)

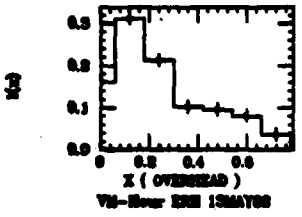


PAGEOUT (Joint)

OVERHEAD (Fund.)

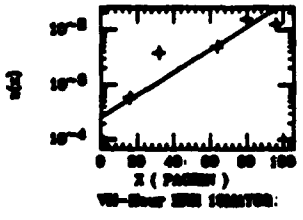


OVERHEAD (Load)



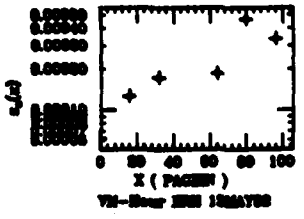
OVERHEAD (Apparent)

PAGEIN (Fund.)

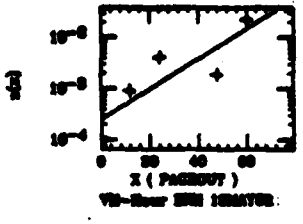


PAGEIN (Load)

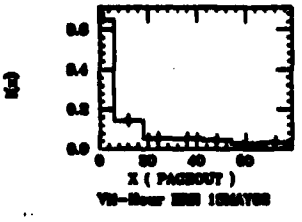
PAGEIN (Apparent)



PAGEOUT (Fund.)



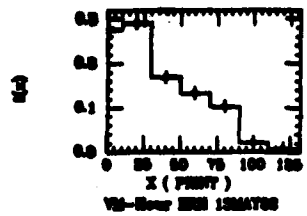
PAGEOUT (Load)



PAGEOUT (Apparent)

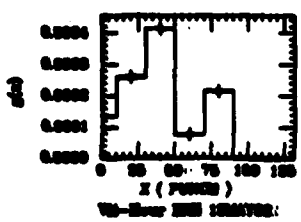
PRINT (Cond.)

PRINT (Load)



PRINT (Joint)

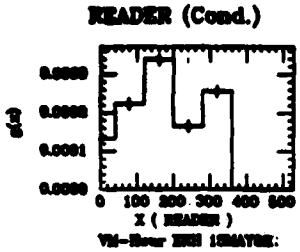
PUNCH (Cond.)



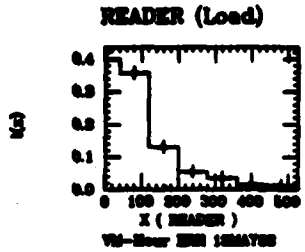
PUNCH (Load)

PUNCH (Joint)

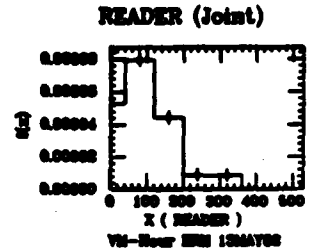
READER (Cond.)



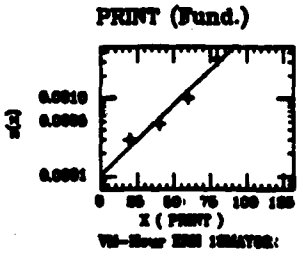
READER (Load)



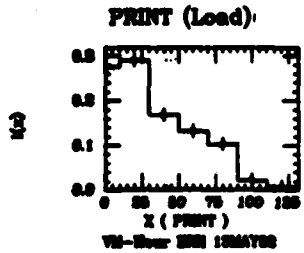
READER (Joint)



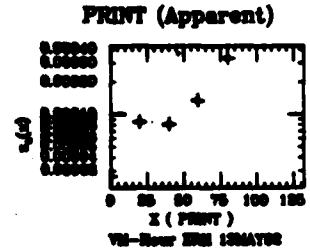
PRINT (Fund.)



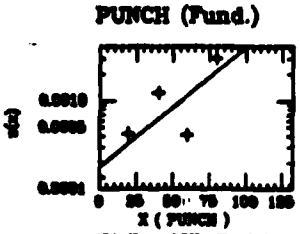
PRINT (Load)



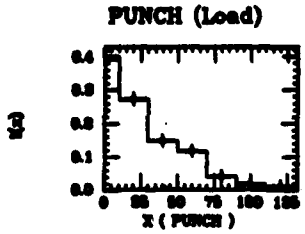
PRINT (Apparent)



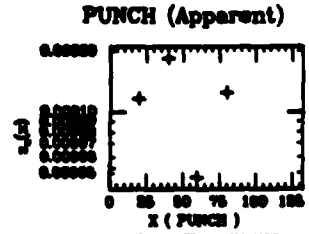
PUNCH (Fund.)



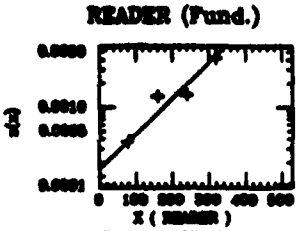
PUNCH (Load)



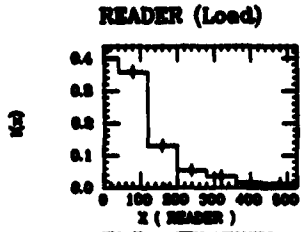
PUNCH (Apparent)



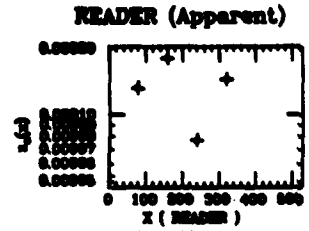
READER (Fund.)

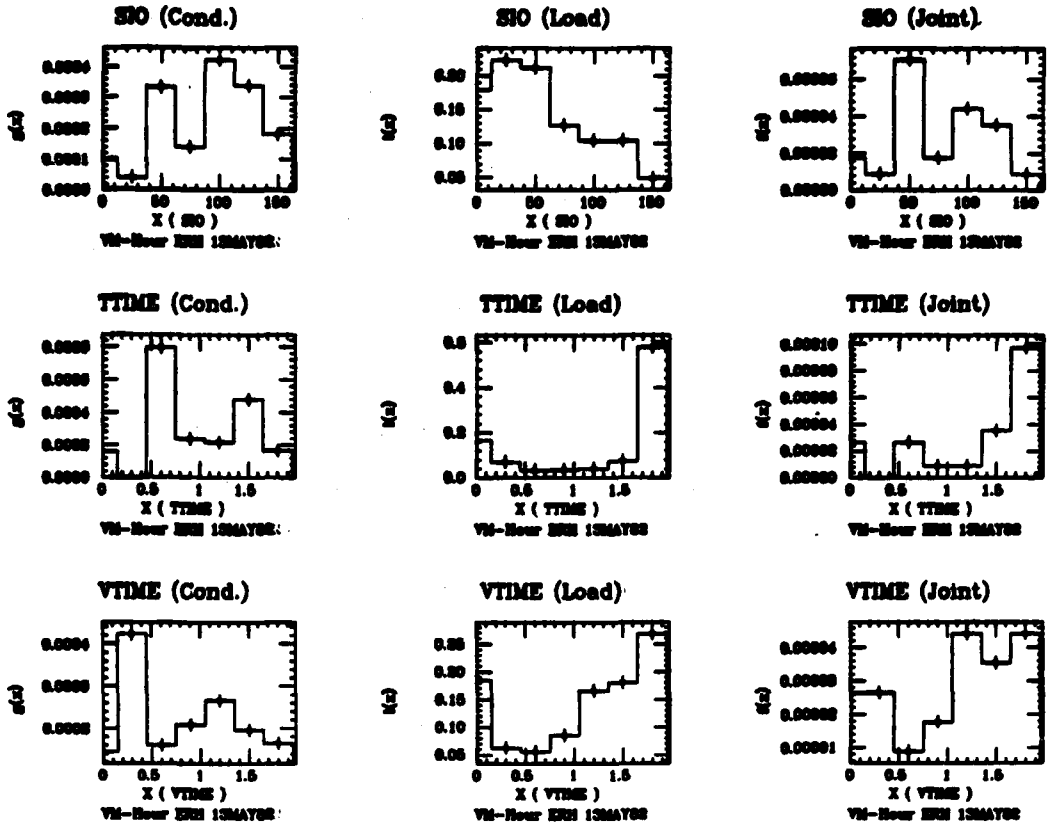


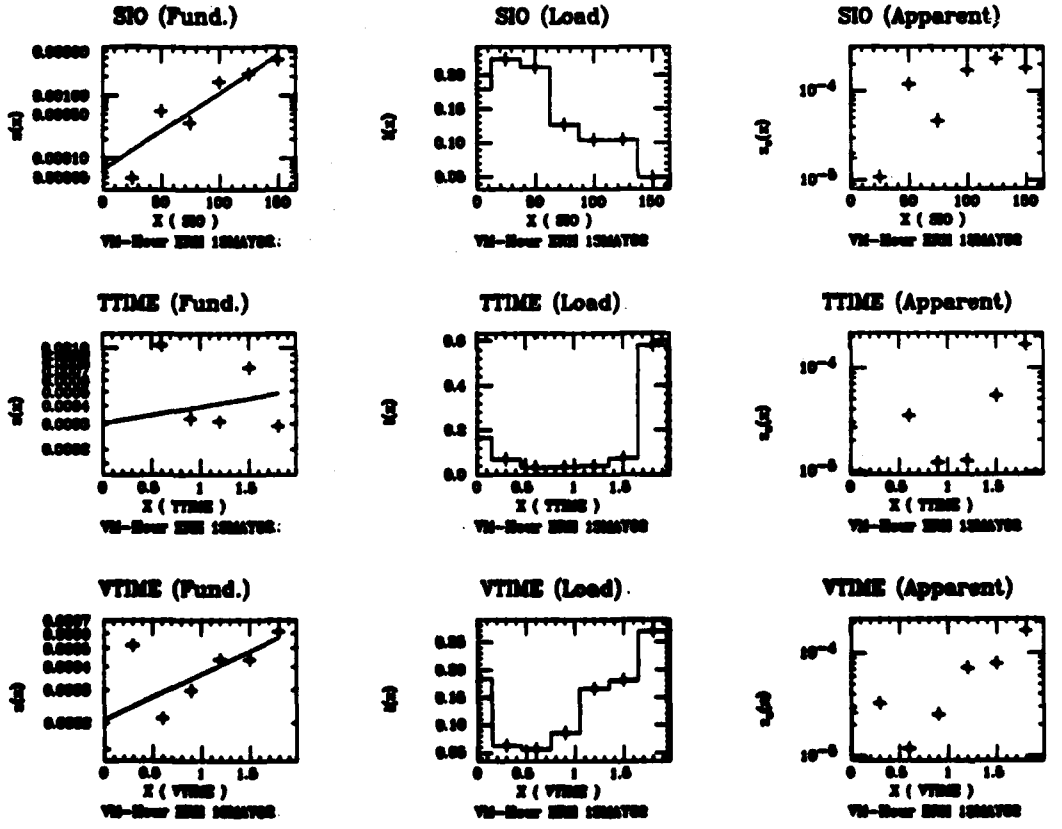
READER (Load)



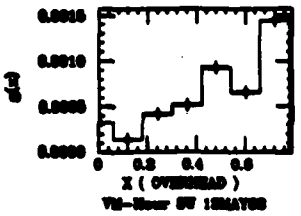
READER (Apparent)



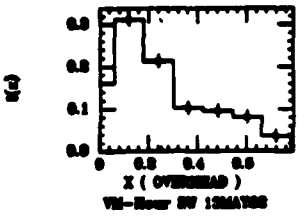

SIO (Cond.)
SIO (Load)
SIO (Joint)
TTIME (Cond.)
TTIME (Load)
TTIME (Joint)
VTIME (Cond.)
VTIME (Load)
VTIME (Joint)
X (SIO)
X (TTIME)
X (VTIME)


SIO (Fund.)
SIO (Load)
SIO (Apparent)
TTIME (Fund.)
TTIME (Load)
TTIME (Apparent)
VTIME (Fund.)
VTIME (Load)
VTIME (Apparent)
X (SIO)
X (TTIME)
X (VTIME)

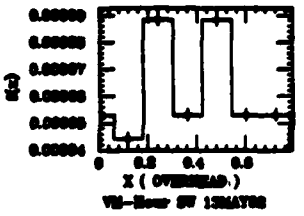
OVERHEAD (Cond.)



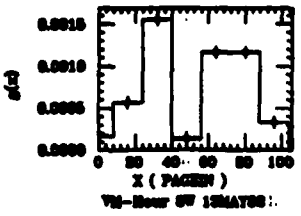
OVERHEAD (Load)



OVERHEAD (Joint)



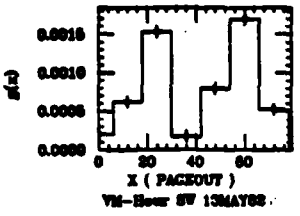
PAGEIN (Cond.)



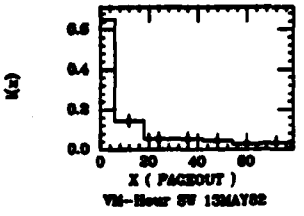
PAGEIN (Load)

PAGEIN (Joint)

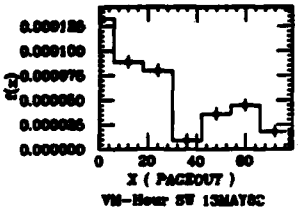
PAGEOUT (Cond.)



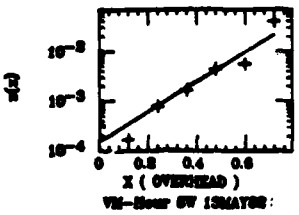
PAGEOUT (Load)



PAGEOUT (Joint)

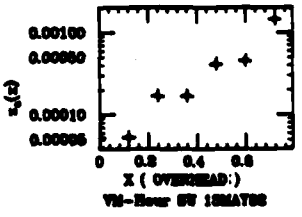


OVERHEAD (Fund.)

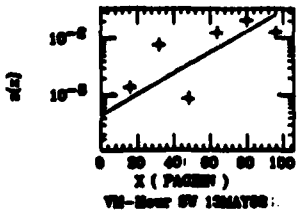


OVERHEAD (Load)

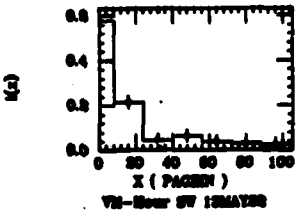
OVERHEAD (Apparent)



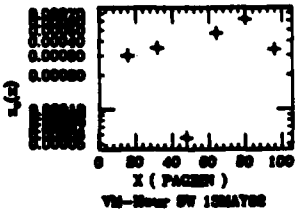
PAGEIN (Fund.)



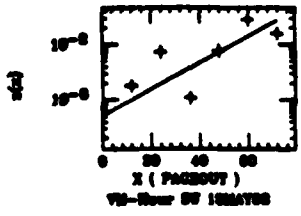
PAGEIN (Load)



PAGEIN (Apparent)



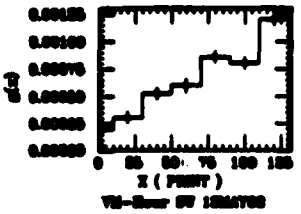
PAGEOUT (Fund.)



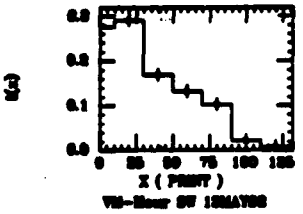
PAGEOUT (Load)

PAGEOUT (Apparent)

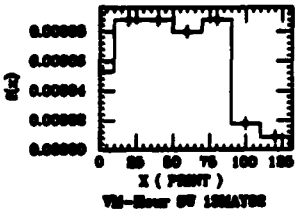
PRINT (Cond.)



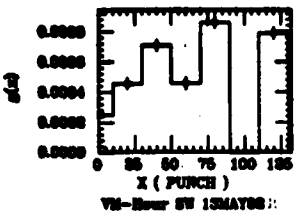
PRINT (Load)



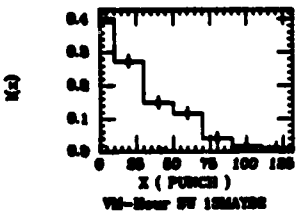
PRINT (Joint)



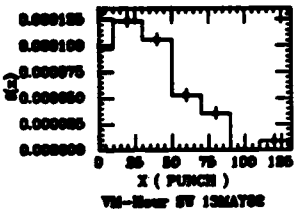
PUNCH (Cond.)



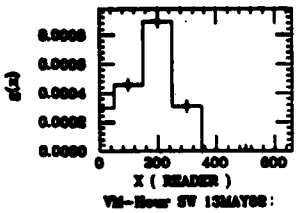
PUNCH (Load)



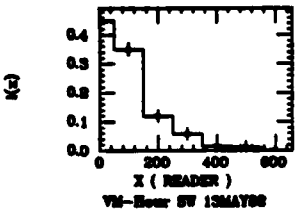
PUNCH (Joint)



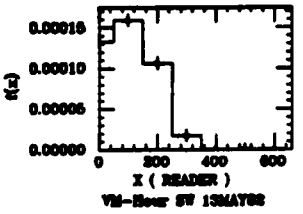
READER (Cond.)



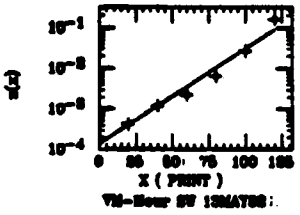
READER (Load)



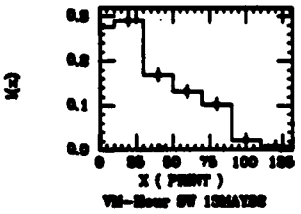
READER (Joint)



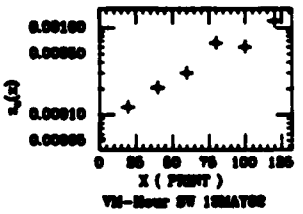
PRINT (Fund.)



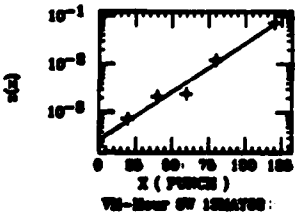
PRINT (Load)



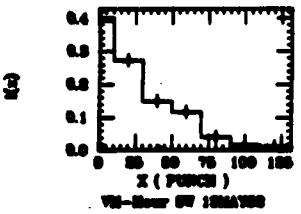
PRINT (Apparent)



PUNCH (Fund.)

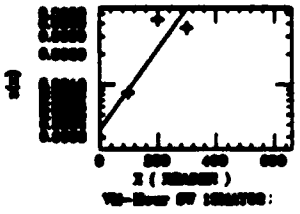


PUNCH (Load)

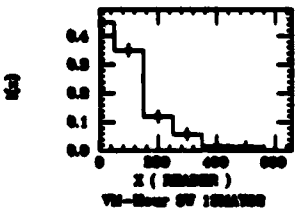


PUNCH (Apparent)

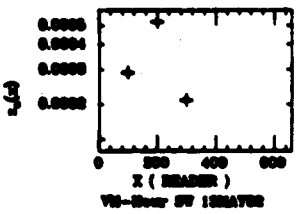
READER (Fund.)



READER (Load)

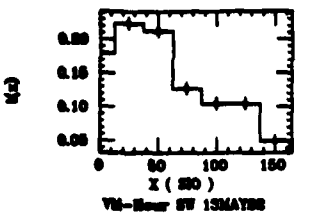


READER (Apparent)

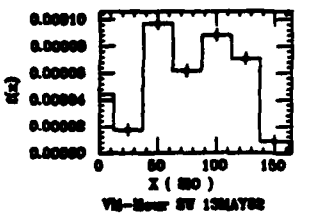


SIO (Cond.)

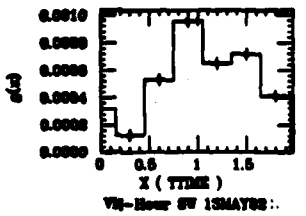
SIO (Load)



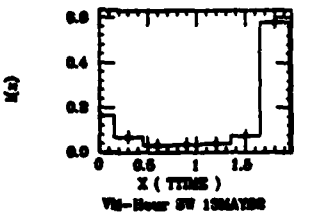
SIO (Joint)



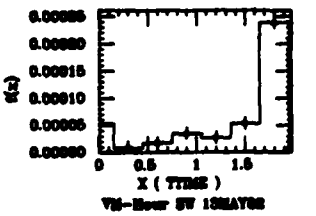
TTIME (Cond.)



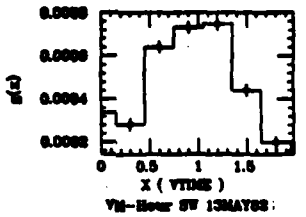
TTIME (Load)



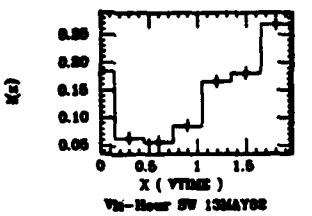
TTIME (Joint)



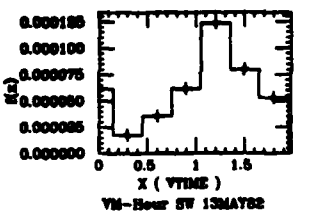
VTIME (Cond.)



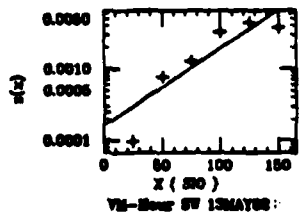
VTIME (Load)



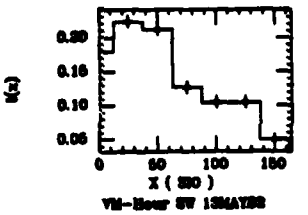
VTIME (Joint)



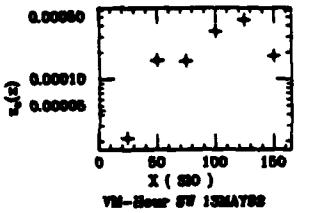
SIO (Fund.)



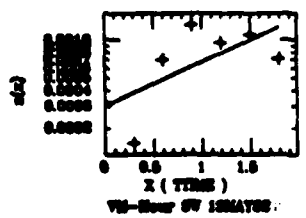
SIO (Load)



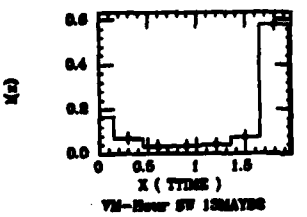
SIO (Apparent)



TTIME (Fund.)

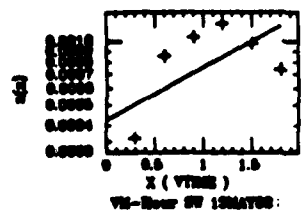


TTIME (Load)

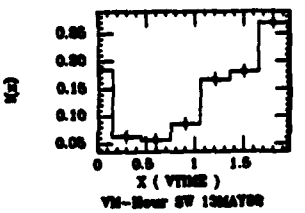


TTIME (Apparent)

VTIME (Fund.)



VTIME (Load)



VTIME (Apparent)